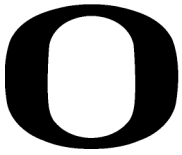

UNIVERSITY OF OREGON

APPLIED INFORMATION MANAGEMENT

Presented to the Interdisciplinary Studies Program:
Applied Information Management and the Graduate School of the University of Oregon
in partial fulfillment of the requirement for the degree of Master of Science

Applying Lean Thinking Principles to Software Development

CAPSTONE REPORT

Ray Tatum
Program Manager
Hewlett-Packard

University of Oregon
Applied Information Management Program

June 2005

722 SW Second Avenue
Suite 230
Portland, OR 97204
(800) 824-2714

Approved by

_____

Dr. Linda F. Ettinger
Academic Director, AIM Program

Abstract

For

Applying Lean Thinking Principles to Software Development

Lean thinking principles, based on the Japanese concept *muda*, have been successfully applied in manufacturing and product development organizations since the 1940s. The software development community can realize similar benefits, with potential to eliminate wasted efforts inherent in the serial and structured traditional software development process. This study defines the seven basic principles of lean thinking (Poppendieck and Poppendieck, 2003), examines how they relate to the software development process and suggests techniques for their application.

Table of Contents

List of Tables

List of Figures

Chapter I. Purpose of the Study

Brief Purpose

Traditional software development has embraced methodologies that are very serial and structured (Kennedy 2003). Another system is available, called “lean thinking”, that provides an alternative to these traditional methods. Lean thinking is defined by Womack and Jones (1996) as "… a way to specify value, line up value-creating actions in the best sequence, conduct these activities without interruptions whenever someone requests them, and perform them more and more effectively." (p.15). According to several lean thinking experts including James Womack (1996), Mary and Tom Poppendieck (2003), and Michael Kennedy (2003), traditional software development creates wasted engineering efforts. These same authors suggest that lean thinking provides ways to eliminate this waste.

The purpose of this study is to provide a set of resources for software development managers regarding the principles of lean thinking (Womack and Jones 1996). The intent of these resources is to (1) examine how these principles apply to software development (Kennedy 2003) (2) in ways that have potential to eliminate wasted efforts inherent in the more traditional software development process.

A review of existing literature (Leedy and Ormrod 2001) is conducted to select published materials regarding lean thinking principles and how they relate to software development. The goal of literature selection is not only to obtain information about the principles of lean thinking, but also how lean thinking can be applied to software development to enable organizations to realize the benefits of lean thinking.

Once the literature resources are collected, they are examined using content analysis with a conceptual analysis approach (CSU Writing Center, 2005). Content analysis is conducted in two phases. Phase One identifies the principles of lean thinking and the principles of traditional software development. Phase Two examines selected case examples in the literature of the application of lean thinking principles, in order to identify how lean thinking processes can be used in the software development process to eliminate wasted efforts inherent in the more traditional software development process.

The results emerging from this content analysis are formatted as a set of resources for software development managers in two different outcomes. The outcome derived from Phase One is presented in two forms: two tables and a set of definitions. The first table provides software development managers with a solid foundation of the principles of lean thinking. Overviews of the principles are presented in the table, including sources cited for each. In addition to the overviews there are detailed definitions of the principles. A second table provides lists lean thinking software development principles, derived from the writings of Kennedy (2003) and the Poppendiecks (2003). The outcome derived from the Phase Two relates lean thinking principles to traditional software development processes. The second outcome is presented in two forms: a narrative and a table. The narration aligns a summary of each lean thinking principle to case examples of the software development processes, selected from the literature. This alignment illustrates lean thinking implementation options in relation to software development in ways that have potential to eliminate wasted efforts. In addition to the narrative, there is a table that lists lean thinking software development principles, derived from the writings of Kennedy (2003) and the Poppendiecks (2004). This table shows which lean thinking

principles can be applied to software development processes and the benefits from the application of the principles, in relation to eliminating waste.

Full Purpose

The purpose of this study is to provide managers who lead software development organizations with an overview of the principles of lean thinking and a set of related resources to enable extension of these principles into the realm of software development. Since lean thinking was originally developed with manufacturing in mind (Womack, Jones and Ross 1991), a discussion on how these principles apply to software development is extrapolated by the researcher, with emphasis on ways to eliminate wasted efforts.

Traditional software development strategies generally follow a very linear process and are required to adhere to a very structured framework (Kennedy 2003). The linear process includes project management practices that cover how to conduct meetings, format reports, define required reports and other defined procedures. Reports can include Gantt Charts, Pert Charts, Work Breakdown Structures, Task Usage, Resource Usage and others (Gray and Larson 2000). When applied to software development, a set of required meetings are defined including project, design review, code walk-through, schedule status and defect review meetings.

Kennedy (2003) outlines the typical five stages of software development as system definition, design specifications, subsystem development, prototyping, and final integration. Typically, a very rigid schedule is created in the early stages of a project. If a defect is identified in any phase of the program and it has to loop back to a previous phase, it can result in a restart from that point and the project schedule can be drastically

impacted (Kennedy 2003). This restart can be due to any number of factors, including a resulting design change, major turmoil in the code base or a change in the project objectives. Adherence to these traditional processes can create waste and reduce an engineer's productivity. On average, as reported by Kennedy (2003), engineers working on projects that followed rigid processes with numerous meetings and reports were only spending 20% of their time in added-value activities, while engineers that followed lean thinking principles were spending 80% of the time on productive tasks.

Another system is available that can be applied to the software development process, called “lean thinking”. Lean thinking provides an alternative to the traditional linear method. Lean thinking is defined by Womack and Jones (1996) as "… a way to specify value, line up value-creating actions in the best sequence, conduct these activities without interruptions whenever someone requests them, and perform them more and more effectively" (p. 15). While lean thinking principles have their foundation in manufacturing, software development organizations should be able to realize benefits when applying them to traditional software development processes (Kennedy 2003). The writings of Kennedy (2003) and Poppendieck and Poppendieck (2003) provide examples of how lean thinking principles can be applied to software development resulting in the elimination of waste that is created by traditional methods.

Principles of Lean Thinking

Lean thinking is based on the concepts of the Toyota Production System pioneered by Taiichi Ohno in the 1940s. There are seven principles of lean thinking, of which eliminating waste is just one. The full set of seven principles is (Poppendieck and Poppendieck, 2003):

1. Eliminate waste.
2. Amplify learning.
3. Decide as late as possible.
4. Deliver as fast as possible.
5. Empower the team.
6. Build integrity in.
7. See the whole.

A brief explanation of each of the principles as relevant to the focus of this study follows. Viewed as a set, these seven principles are guiding ideas and insights about lean thinking. While the principles are generic, the application of processes must be tailored to a specific environment (Poppendieck and Poppendieck 2003).

Eliminate waste. The core principle of lean thinking that all others are based on is eliminating waste. Once somebody has developed the skill of identifying waste, they have started on their path to implementing lean thinking (Poppendieck and Poppendieck 2003). Ohno defines this waste as anything that does not create value for the customer (Poppendieck and Poppendieck 2003). Ohno (1988) uses the Japanese word *muda* to identify waste and defines it as:

> defects (in products), overproduction of goods not needed, inventories of goods awaiting further processing or consumption, unnecessary processing, unnecessary movement (of people), unnecessary transport (of goods) and waiting (by employees for process equipment to finish its work or on an upstream activity) (pp. 19-20).

Womack and Jones (1996) provide the following examples of muda:

- Mistakes that require rectification

- Production of items that nobody wants
- Processing steps that are not needed
- Movement of resources without purpose
- People waiting for deliverables
- Defective releases
- Goods and services which don't meet the needs of the customer

Amplify learning. At the center of this principle is the concept to learn from past experience (Womack and Jones 1996). While this concept seems obvious, many times when a problem is solved the solution is not documented. If the issue is seen again the solution has to be reconstructed, resulting in waste (Poppendieck and Poppendieck 2003). Processes need to be put into place to capture and make experiences available in the future.

Decide as late as possible. Traditionally software requirements are established and finalized before any development begins. This approach tends to force designers to establish a depth focus on the project instead a breadth focus (Poppendieck and Poppendieck 2003). This type of approach is what leads to looping back into a previous stage that was previously mentioned. If design decisions are made earlier rather than later, options for solving problems that are identified later in the development process are limited (Womack and Jones 1996).

The principle of concurrent development or set-based design provides a way to begin with a breadth approach and then when appropriate move to a depth view. The core principle behind concurrent development and set-based design is to work on several different solutions at the same time and as problems surface, eliminate ones that do not meet the needs of the project. The final decision as to which solution to use is prolonged

to the last possible moment, either when only one solution is obviously feasible or when a decision has to be made because other parts of the project are waiting on the solution (Poppendieck and Poppendieck 2003).

Deliver as fast as possible. The more frequent the deliveries are in a project, the sooner issues can be identified (Poppendieck and Poppendieck 2003). If the development team is on a schedule to deliver releases to test once a month, it will take two months for any fixes to be seen in future releases. When this schedule is modified so that the releases are available on a weekly cycle, then fixes are seen in a maximum of two weeks (Poppendieck and Poppendieck 2003).

Empower the team. This principle addresses the organizational level that decisions are made. Decisions need to be made at the lowest level possible in order to allow the person with the best knowledge of the issue to make the decision (Womack, Jones and Ross 1991). There is a tendency for managers to want to make the final decision since they are the ones who are ultimately responsible. They need to learn to trust the people that have the knowledge to make the right decisions and focus their attention on issues that belong to them (Poppendieck and Poppendieck 2003). Compare this to the way that planning takes place in the Marine Corps. When planning and executing a campaign, high-ranking officers and intelligence groups plan the strategy and the big picture of the campaign. This includes the high-level goals and responsibilities. Once the mission is underway and issues arise on the battlefield, the officers and men on site make the decisions on how to adjust to address the issues.

Build integrity in. Two types of integrity are defined in this principle by Poppendieck and Poppendieck (2003) -- perceived and conceptual. They write:

> Perceived integrity means that the totality of the product achieves a balance of function, usability, reliability and economy that delights customers. Conceptual integrity means that the system's central concepts work together as a smooth cohesive whole. (p. 125).

The Poppendiecks (2003) further define perceived integrity as what the customer thinks the integrity is. It can be influenced by marketing, sales, and other factors that have nothing to do with the true quality of a product. Conceptual integrity is based more on how well the product is designed and developed. When you combine conceptual and perceived integrity the result is what is known as product integrity.

A study at Harvard Business School by Kim Clark (Clark and Fujimoto1994) shows that companies that value product integrity and make it a part of the development methodology consistently deliver superior products. Lean thinking takes this concept to heart and says that product integrity should influence everything you do when making decisions about the development of a product (Ward 2002).

See the whole. In this principle, Poppendieck and Poppendieck (2003) describe a system as not the sum of the parts but the product of the interactions. One of the challenges for developers is that while they are working on small pieces of the solution they must never lose sight of the overall picture of the product. When developers lose sight of the overall picture, problems arise around the interaction points of the different parts. (Poppendieck and Poppendieck 2003).

A literature review was selected as the research method for this study (Leedy and Ormand, 2001). The search for literature begins with the reading of *Lean Thinking Banish Waste and Create Wealth In Your Corporation* (Womack and Jones 1996) and *Product Development for the Lean Enterprise* (Kennedy 2003). From these readings,

additional resources are identified that provide a base knowledge of the principles of lean thinking. For the second stage of data collection, the writings of Kennedy and the Poppendiecks are reviewed for a foundation of how to apply lean thinking to software development.

Conceptual Analysis is chosen as the strategy to perform the content analysis of the reviewed literature (CSU Writing Center 2005). The CSU Writing Center guide identifies eight steps in performing conceptual analysis. They are:

1. Decide the level of analysis.
2. Decide how many concepts to code for.
3. Decide whether to code for existence or frequency of a concept.
4. Decide on how you will distinguish among concepts.
5. Develop rules for coding your texts.
6. Decide what to do with “irrelevant” information.
7. Code the texts.
8. Analyze your results.

These steps are used to identify writings with the theme of "lean thinking', "software development" and "waste". Rules are established to further identify writings that addressed all three of these themes. Additionally these steps allow the data to be analyzed in chunks of manageable data and determine if relevant information is contained. Phase One of the content analysis examines writings that address lean thinking principles as defined by Womack and Jones (1996) and Kennedy (2003). How the principles are being applied or what type of organization is applying them is not taken into consideration. Also included in this phase is an overview of traditional software development principles.

Phase Two examines case study reviews that address how lean thinking principles have been applied to software development organizations

The result of each phase of the content analysis is presented in separate outcomes. The first provides a list of lean thinking principles, framed for the software development manager as a foundation to understand lean thinking principles. The outcome includes detailed explanations of each of the principles with cited references if further information is required and a table that summarizes each of the principles in a one or two sentence definition.

Outcomes of the second phase include examples of how lean thinking principles can be applied to traditional software development procedures. Included in this outcome are examples of how the principles have been applied in real-life experiences, with a summary of the resulting impacts from the implementation of lean thinking principles. A three-column table summarizes the data collected during the second phase of data analysis. The first column contains a traditional software development process, the second identifies a lean thinking principle that can be applied to it, and the final column lists the resulting benefit related to eliminating waste.

Limitations to the Research

Lean thinking has its roots in the automobile design and manufacturing industry, more specifically, Toyota Corporation (Womack, Jones and Ross 1991). Since the focus of this paper is the application of lean thinking on software development, how lean thinking pertains to manufacturing was not addressed specifically. Reference to Toyota and manufacturing is in the context of background when discussing the principles of lean thinking.

While the foundation upon which lean thinking is based started at Toyota in the early 1940s (Poppendieck and Poppendieck 2003), it was not recognized as a viable approach to design and development until the early 1990s. The industry was introduced to lean thinking by books like *The Machine that Changed the World* (1991) by James Womack, Daniel Jones and David Ross.

While traditional software development practices are not the focus of this study, they need to be reviewed and examined for context. Since they are contained as a reference point, only one resource is utilized, *Project Management The Managerial Process* (Gray and Larson 2000). This book is picked as the single source for traditional software development practices based on the reputation of the authors. They are both long time members of the Portland, Oregon chapter of the Project Management Institute. Both Gray and Larson have published numerous articles concerning management and project management. Their consulting activities focus on project management (Gray and Larson 2000).

Literature with focus pertaining to software development and lean thinking did not appear until late in the 20th century. Works applying lean thinking to software development are found post-1995, while works with focus on case studies and emphasis on best practices are from the early 2000s. In addition the majority of this literature is from two authors, Mary and Tom Poppendieck. They are recognized by other experts in the field of Lean Thinking. Allan Ward (2002), Michael Kennedy (2003), Jerry Kilpatrick (2003) and Mark Windholtz (2003) reference them in their works on the subject. The majority of the additional literature located on this part of the subject either supported the Poppendiecks or a review of their publications.

The majority of the literature located on the subject of lean thinking is in the form of case study reviews and examples of lean thinking principles in the industry. Analysis of these case study reviews and examples provide insight into practices that have been successful, and can be expressed as suggested techniques for managers to aid them in their own transition.

Problem Area

Traditional software development has embraced methodologies that are very serial and structured. Requirements and specifications are developed down to the minute detail before coding even begins, as well as multiple teams with very narrow areas of responsibility and structured reporting guidelines (Poppendieck and Poppendieck 2003). The software product life cycle is defined by Gray and Larson (2000) as a process that sequentially passes through definition, planning, execution and delivery phases. The activities that typically occur in each phase are listed in Table 1: Software Product Life Cycle below (from Gray and Larson, 2000).

Definition	Planning	Execution	Delivery
• Goals • Specifications • Tasks • Responsibilities • Teams	• Schedules • Budgets • Resources • Risks • Staffing	• Status Reports • Changes • Quality • Forecasts	• Train customers • Transfer documents • Release resources • Reassign staff • Lessons learned

Table 1: Software Product Life Cycle

One of the cornerstones of the traditional project lifecycle is that you must complete one phase of the cycle before you can move onto the next. This process falls apart when a defect or issue is identified in the execution phase that requires a change in

the specifications. If you are following the traditional paradigm to the letter, you must then go back to the definition phase and start over. While the second pass through will be shorter than the original, it may still significantly impact the schedule (Kennedy, 2003).

According to Moore (1991), traditional software methodologies work fine when an organization is in the Tornado phase – that time when an organization is working in technology that is new. When new aspects of a technology are developing rapidly, the consumer is willing to pay a premium price for the technology (Moore 1991). In this phase, sales are booming, profit margins are high and budgets are unlimited, resulting in an environment of 'just do what you have to do to get product out'. Methods used in the Tornado stage produce waste that is considered to be acceptable due to the high profits and 'just get it out the door' mentality. This waste is identified by Poppendieck and Poppendieck (2003) as anything that does not create value as perceived by the customer.

Moore (1991) contends that as the technology matures, competitors enter the market and consumers start demanding a more competitive price point. This moves the business into the Main Street phase. In this phase sales are leveling off, profit margins are reduced, and budgets are tight. Managers need to look for ways to reduce development waste when they have entered the Main Street phase so that software development becomes a lean, mean, producing machine (Moore 1991).

Kennedy (2003) and Womack and Jones (1996) each suggest that one potential benefit of utilizing lean thinking principles in software development is that it may provide an excellent way to reduce the development waste. The waste elimination will result in shortened development time and reduced development cost (Kennedy 2003). Lean thinking is an approach that has been proven successful in several organizations (Womack 1996) for eliminating "waste".

The following comparison of two similar projects provided by Poppendieck and Poppendieck (2003) demonstrates one example of how these methodologies can affect the outcome of a project:

> Jim Johnson, chairman of the Standish Group, told an attentive audience the story of how Florida and Minnesota each developed its Statewide Automated Child Welfare Information Systems (SACWIS). In Florida, system development started in 1990 and was estimated to take 8 years and to cost $32 million. As Johnson spoke in 2002, Florida had spent $170 million and the system was estimated to be completed in 2005 at the cost of $230 million. Meanwhile, Minnesota began developing essentially the same system in 1999 and completed it in early 2000 at the cost of 1.1 million. That's a productivity difference of over 200:1. Johnson credited Minnesota's success to a standardized infrastructure, minimized requirements, and a team of eight capable people. (Johnson 2002 p. xxi).

While there is no magic or silver bullet to ensure similar results when applying these types of principles, there are some proven practices that enable higher performance and waste elimination (Poppendieck and Poppendieck 2003). These practices, known as the Toyota Lean Production System, have their roots in automotive production and were developed by Taiichi Ohno working at Toyota in the 1940s. This system results in less direct labor, minimizes work-in-process, reduces parts inventory and results in fewer defects. After realizing the benefits from these practices in their production organization, Toyota leveraged this system into their product development teams and titled this new system Lean Product Development. This system is aimed at elimination of wasted effort and the application of other lean production principles to product development (Kennedy

2003). The phrase lean thinking has become associated with the core principles of both lean production and lean development (Womack and Jones 1996).

Chapter II. Review of References

The review of references provides an annotation of the primary resources used for this paper. These and all other resources are listed in the References. Each annotation addresses the following three issues: a summary of the resource's content that is pertinent to this study, which sections of the study the resource is relevant to, and the criteria used to select the resource.

CSU Writing Center (2005). *Introduction to Content Analysis 2005.* Retrieved March 2005 from: http://writing.colostate.edu/references/research/content/.

This web-based source provides an overview of content analysis and how to apply it to a research project. It is maintained by Mike Palmquist and students from the Department of English at Colorado State University. This source was used as a guideline on how to conduct the content analysis for this paper. It was chosen because of its association with a credentialed institution and Palmquist's reputation in the field of content analysis.

Gray, C. and Larson, E. (2000). *Project Management The Managerial Process* Boston: Irwin McGraw-Hill.

Gray and Larson's book provides a basic understanding of traditional software development principles. Since these principles are not the central focus of this paper, it is the only source used for these principles. Gray and Larson are used to explain traditional software development practices in the Purpose and Problem areas of this study. As stated in the Limitations section of this paper, it was chosen on the reputation of the authors and because it is the primary text for project management in the University of Oregon AIM Masters program.

Kennedy M. (2003). *Product Development for the Lean Enterprise* Richmond: The Oaklea Press.

This book provides a review of the process a product design organization might go through in making the decision to move toward lean development. It provides a view of what is lacking in traditional product development and how lean principles can address this gap**.** The book is highly recommended by lean thinking expert Allen Ward as an introduction to lean thinking and how it helps an organization develop the framework for lean thinking. It was chosen based on the recommendation of Dr. Ward who has been a consultant on lean thinking in product development for Hewlett-Packard. Kennedy's book is quoted in the Purpose and Problem areas of this study to support what is wrong with current software development techniques and how they can benefit from lean thinking.

Moore, G. (1991). *Crossing the Chasm* New York: Harper Business.

Geoffrey Moore's book describes the need to tailor development and marketing strategies for high tech products more carefully than other industries. It provides examples of how high-tech organizations can make the transition from the Tornado phase of a product into the Main Street phase. Lean thinking is one of the techniques he discusses.

Moore is quoted in the Purpose area of this study to show some of the issues with product development in the high-tech industry and why some corporations have incorporated lean thinking into their day-to-day processes.

This book was selected because of Moore's in-depth analysis of the issues with development in the high-tech industry and how lean thinking can address these issues.

Ohno, T. (1988). *The Toyota Production System: Beyond Large Scale Production* Portland: Productivity Press.

Taiichi Ohno was inventor of Just-In-Time (JIT) manufacturing at Toyota. JIT was the foundation upon which lean thinking was built. Ohno's principles and guidelines are still considered to be the cornerstones of lean thinking and this resource is selected on this basis. Ohno is recognized as the father of lean thinking by experts in the field including Womack (1991), Poppendieck (2003) and Kilpatrick (2003).

This resource also provides a definition of waste, one of the foundations of lean thinking and a look at the early beginnings of lean thinking. Quotes and data from the resource are utilized in the Problem Area and Analysis of Data sections to aid in the explanation of the origin of lean development.

Poppendieck, M and Poppendieck T.

Mary and Tom Poppendieck are considered to be two of the leading experts in the application of lean thinking to software development. They own and manage Poppendieck, LLC that is a company dedicated to bringing lean thinking to software development. Both Tom and Mary instruct courses on lean software development and agile design thinking. The Poppendiecks are quoted in the Purpose, Problem and Analysis of Data areas of this study to aid in the definition of lean thinking and how it can be applied to software development.

Several works of the Poppendiecks are selected as key references to support this study because of their insight and knowledge of lean software development. This is seen by the volume of the publication, the diversity of where it is published and other experts' references to them in their own publications.

Poppendieck, M. and Poppendieck T. (2003). *Lean Software Development An Agile Toolkit* Boston: Addison Wesley.

In this resource, Mary and Tom Poppendieck identify seven lean principles and provide techniques for implementing them in a software development environment. This resource is used as a foundation to identify how to realize improvements in software development organizations from the implementation of lean thinking. Quotations from this publication are used in the Purpose and Analysis of Data sections of this paper to support the theory that lean thinking principles can be beneficial to software development. It was also utilized in the Problem Area to emphasize the issues with current software development principles.

In addition to the reasons for selecting work by the Poppendiecks already mentioned in the introduction of these authors, this book was selected because of its focus on software development and lean thinking principles. This work is referenced by numerous articles on the web that address lean thinking as it applies to software development.

Poppendieck, M. (2002). *Principles of Lean Thinking.* Retrieved March, 2005 from: http://www.poppendieck.com/papers/LeanThinking.pdf.

Mary Poppendieck provides an introduction to lean thinking in this resource. In respect to this study it provides an excellent mapping of lean thinking principles to software development. Quotes from this resource are used to support the application of lean thinking principles to software development in the Analysis of Data area of the study. It was chosen because of the emphasis on lean thinking and software development as well as the reputation of the author as an expert in the field of lean software development.

University of Oregon (2005). *Evaluating Information on the World Wide Web Service of University Libraries*. Retrieved March, 2005 from: http://libweb.uoregon.edu/guides/searchweb/evaluating.html

A web-based resource maintained by the University of Oregon that provides guidelines for evaluating information on the World Wide Web. It provides criteria for researchers to evaluate web articles to determine if the information is credible and viable for their project.

Ward A. (2002). *The Lean Development Skills Book* Ann Arobt: Dollar Bill Books

This publication is a pocket book written by Allen Ward that provides managers a quick reference on how to apply lean development into their organizations. It provides concise definitions of lean thinking principles. The information provided by this publication was used in the Purpose and Analysis of Data areas of this study to provide clarification of lean thinking. Ward is recognized by experts in the field of lean thinking, such as James Womack, to be the leading U.S. authority on Toyota's product development process. The author attended a talk by Dr. Ward presented to development

managers at Hewlett-Packard in 2004. The ideas and concepts presented in this talk formed the impetus for this study.

Womack, J and Jones D. (1996). *Lean Thinking Banish Waste and Create Wealth In Your Corporation* New York: The Free Press.

In addition to another view on Toyota lean thinking principles, Womack and Jones provide case study reviews of these principles as they are applied in the automotive, aerospace and other manufacturing industries. While the information is not focused on software development, it does provide examples and arguments of how lean thinking can improve productivity. Information from this resource provides quotes that aid in the development of the definition of lean thinking and are quoted in the Purpose, d Problem and Analysis of Data areas of this study. Literature, including The Machine That Changed the World, authored by Womack and his associates. Womack is the founder and president of the Lean Enterprise Institute and has authored or co-authored several publications on lean thinking over the past twenty years including *Lean Thinking Banish Waster and Create Wealth In Your Corporation (1990), The Machine That Changed the World, Lean Thinking (1986), Lean Thinking (1996), From Lean Production To The Lean Enterprise (1994) and Beyond Toyota: How to Root Out Waste and Pursue Perfection (1996).* Womack is also referenced by Kennedy, Ward and the Poppendiecks in their publications.

Womack, J, Jones, D. and Ross, D. (1991). *The Machine That Changed the World* Harper Perennial.

Womack, Jones and Ross explain lean production and discuss its implications for development organizations. The explanation is based on a five-year study of the worldwide auto industry and provides insights on how lean production has improved productivity. Quotes from this resource are used in the Purpose and Analysis of Data areas of this study to support the basic principles of lean thinking and their benefits. It was selected because of Womack's reputation that is addressed in the review of his work *Lean Thinking Banish Waste and Create Wealth In Your Corporation*.

Chapter III. Method

Data Collection

Literature review (Leedy and Ormrod 2001) is the method of study used in this work. This technique reviews the works of others and develops a view or interpretation of the data. The researcher began the search for resources on lean thinking with a conversation with colleagues at Hewlett-Packard who have been active in investigations around lean thinking and software development. They suggested Moore (1991), Womack (1996) and Reinersten. (1997). An initial review of these books provided key words and reference points for a literature search.

For articles from the WEB the Google search engine is used with lean thinking, lean development and software development as key words. Additional limitations are applied to searches to narrow the results. Results with references to Amazon were removed to eliminate WEB pages that are for ordering books. A restriction to limit the results to articles written in English is also applied. A final restriction is placed to limit results in which the key phrases are part of the title. The refined searches and their results are listed in Table 2: Search Strategy.

Search String	Initial Search	- Amazon	English Only	Title Only
"Lean Thinking"	72,900	59,100	48,100	3,840
"Lean Thinking" + "Software Development"	4640	4150	4010	25
"Lean Development"	5,400	4,770	3,730	114
"Lean Development" + "Software Development"	882	739	992	19

Table 2: Search Strategy

The resulting 44 articles are evaluated for authority, objectivity, accuracy and currency as outlined by the University of Oregon Libraries (2005). Authority addresses who the author is, what are their credentials and institutions with which they are affiliated. Objectivity tries to identify the purpose of the paper, if the goals of the paper are stated, and if the paper is designed to educate or if it is a soapbox for the author and if any affiliation bias is evident in the paper. Accuracy is used to evaluate the paper in respect to the grammar, formatting, layout and completeness. Finally currency looks at the creation/revision dates, do all of the links work and if the paper has been kept current. This evaluation results in 32 web articles for data analysis.

The initial review of the works of Kennedy (2003), Moore (1991) and Womack (1996) also identifies several books for review such as Beck (2000), Reinerstein (1997), Poppendieck and Poppendieck (2003), Cusummano (1998) and Ward (2002) on the subject of lean thinking and it's application to software development. A Web page at http://www.poppendieck.com/overview.htm is managed by Mary and Tom Poppendieck and provides resources addressing lean thinking and software development and it is used as a resource in this area.

Data Analysis

Conceptual Analysis is chosen as the strategy to perform the content analysis of the selected literature (CSU Writing Center 2005). The CSU Writing Center guide identifies eight steps in performing conceptual analysis. The data analysis is performed in two phases. During Phase One, data is analyzed that pertains to lean thinking principles in general. These steps are applied in the following manner.

1. Decide the level of analysis. – In this step it is decided to code for the occurrence of the phrase lean thinking.
2. Decide how many concepts to code for. – With the focus of this phase being the principles of lean thinking without reference to any type of organization, the analysis is performed using the single concept of lean thinking.
3. Decide whether to code for existence or frequency of a concept. – Since the analysis is looking for literature pertaining to the principles of lean thinking, existence of the term and related concepts is coded for instead of frequency.
4. Decide on how you will distinguish among concepts. – Lean thinking principles are additionally referred to as lean development; therefore, both phrases are coded for to identify the desired concepts.
5. Develop rules for coding your texts. – Coding is performed by identify which lean thinking principle, as identified by Poppendieck and Poppendieck (2003), is referred to in an article.
6. Decide what to do with “irrelevant” information. – References to how lean thinking is implemented in specific organization are ignored since the researcher is looking for general information

7. Code the texts. – The coding of the text is performed by keeping notes from the readings of the sources in a Microsoft Word document. These notes are coded with references to appropriate principle, source and page.
8. Analyze your results. – Analysis of the data is performed by examining the different views obtained from the selected literature and then developing a combined description of the lean thinking principles.

In Phase Two of the content analysis, data is analyzed that examines how lean thinking principles can be applied to software development.

1. Decide the level of analysis. – In this step it is decided to code for the occurrence of both lean thinking and software development
2. Decide how many concepts to code for. – With the focus of this phase being how the principles of lean thinking can be applied to software development the analysis is performed using both the concepts of lean thinking and software development.
3. Decide whether to code for existence or frequency of a concept. – Again, in this step occurrence is code for existence instead of frequency.
4. Decide on how you will distinguish among concepts. – Lean thinking principles are additionally referred to as lean development and thus both phrases are coded for to identify the desired concepts. In addition, software development is also referred to as software project management and software program management so all three of these terms will need to be considered while reviewing data.
5. Develop rules for coding your texts. – Coding is performed by identifying which of the lean thinking principles (Poppendieck and Poppendieck 2003) is referred to

in the literature and which software development practices it is applied to (Gray and Larson 2000).

6. Decide what to do with "irrelevant" information. – References to other paradigms besides lean thinking and how they apply to software development are ignored since that is not the focus of this paper.
7. Code the texts. – The coding of the text is performed by keeping notes from the readings of the sources in a Microsoft Word document. These notes are coded to reference to appropriate principle, source and page.
8. Analyze your results. – Analysis of the data is performed by examining the different ways lean thinking has been, or can be, applied to traditional software development processes. Then a combined description is developed.

Data Presentation

The results of each phase of the content analysis are presented in separate outcomes. The first result provides a list of lean thinking principles, which is then framed for the software development manager (see Table 4: Lean Thinking Principles and Their Components).

This outcome provides detailed explanations of each of the principles with cited references if further information is required. And while specific examples of the implementation of these principles are not provided, examples of general usage are included. In addition to the detailed explanation there is a table that summarizes each of the principles in a one or two sentence definition.

Results of the second phase of content analysis are framed into a table (See Table 5: Lean Thinking Principles and Processes), presenting examples of how lean thinking

principles can be applied to traditional software development procedures. Included in this outcome are examples of how the principles have been applied in real-life experiences, with the resulting impacts. These examples are taken from case study reports and other references to lean thinking and software development in industry. The three-column table contains a traditional software development process in the first column, a lean thinking principle that can be applied to it in the second column, and a description of the resulting benefit related to eliminating waste in the final column.

Chapter IV. Analysis of Data

Data analysis is conducted in two phases as describe in the Data Analysis section of the Method chapter. Thirty-two sources are selected using the criteria outlined by the University of Oregon Libraries (2005). This criterion is authority, objectivity, accuracy and currency. The analysis is conducted using conceptual analysis as described by the CSU Writing Center (2005). Conceptual analysis is composed of eight steps and the application of these steps to this study is explained in the Data Analysis section in Chapter III. Method.

Phase one of data analysis is aimed at locating material that provides a detailed definition of lean thinking in general. Jerry Kilpatrick (2003) defines lean thinking as:

> A systematic approach to identifying and eliminated waste through continuous improvement, flowing the product at the pull of the customer in pursuit of perfection. (p. 1)

While this definition provides a general overview, a much more detailed definition is presented in the Principles of Lean Thinking section of this chapter, as a result of the data analysis. Also presented in this section is additional explication of the core principles of lean thinking. Mary and Tom Poppendieck list seven principles in their book *Lean Software Development An Agile Toolkit* (2003). A set of ten principles is presented in Mary Poppendieck's paper entitled *Lean Programming* (2001). Other experts in the field provide both lists of seven (Kilpatrick 2003) and ten principles (Womack and Jones 1996). While the lists are different in the way they are presented and seldom match exactly, the list of seven principles provided by the Poppendiecks (2003) and explained in the Full Purpose Section in Chapter 1: Purpose of the Study, is representative of all of the

lists. The longer lists just break some of the principles down into two. For clarification, the list use in this study is repeated below (Poppendieck and Poppendieck, 2003):

1. Eliminate waste
2. Amplify learning
3. Decide as late as possible
4. Deliver as fast as possible
5. Empower the team
6. Build integrity in
7. See the whole

Phase two of data analysis is focused in providing the reader with processes that can be utilized in applying lean thinking principles to software development. While there are definite differences between the types of production systems from which the principles of lean thinking sprout and software development, there are enough similarities that enable software development organizations to benefit from lean thinking (Poppendieck and Poppendieck 2003). When this application occurs it is known as Lean Software Development (LSD) Windholtz (2003). Windholtz (2003) defines LSD as a system "…to develop software in one-third of the time, with one-third the budget and with one-third the defect rate."

The Lean Thinking and Software Development section of this chapter provide techniques for applying lean thinking to software development. Also provided is a demonstration of how lean thinking principles map into the activities associated with software development.

Principles of Lean Thinking

The first phase of data analysis in this study is the coding of sources for lean thinking principles in general. A list of words, terms and phrases that relate to lean thinking principles is created from the book *Lean Software Development An Agile Toolkit*

(Poppendieck and Poppendieck, 2003). These terms and phrases are listed in Table 3: Phase One Coding Terms/Phrases.

Coding Terms/Phrases		
Amplify Learning	Extreme Programming (XP)	Queue Sizing
Batch Sizing	Just In Time Development	Reusable Knowledge
Build Integrity In	Knowledge Based Development	See The Whole
Conceptual Integrity	Knowledge Centers	Set-Based Design
Concurrent Development	Late Decision	Sub-optimization
Customer Value	Lean Development	System Thinking
Cycle Time	Lean Manufacturing	Takt Time
Fast Delivery	Lean Thinking	Total Productive Maintenance
Document Learnings	Muda	Trade-off Curves
Eliminate Waste	Non Value Added Processing	Value Stream Maps
Empower The Team	Perceived Integrity	Wasted Steps

Table 3: Phase One Coding Terms/Phrases

After the list is created, seventeen sources are reviewed for the occurrence of these terms or phrases and recorded in a separate form for each source. References to how a specific organization implemented lean thinking are ignored since the goal is to obtain a definition of lean thinking. The format used to record the coding results is located in Appendix B – Phase One Coding Form. In addition to this coding of the existence of the terms, each source and specific page number is notated for further reference (see Appendix C: Phase One Coding Results – Terms that Define Lean Thinking).

Additional analysis is performed to identify similar ways of identifying the same lean thinking principle. Similar data is aligned with a primary category according to which lean thinking principle it represents. The resulting reorganized data is listed in Table 4: Lean Thinking Principles and Their Components. This table provides a mapping of the terms and phrases used in the coding of the data to the principles of lean thinking. It provides some insight into just what each of these principles means.

Lean Thinking Principles	Components
1. Eliminate waste	• Muda • Non Value Added Processing • Value Stream Maps • Wasted Steps
2. Amplify learning	• Concurrent Development • Document Learning • Knowledge Based Development • Knowledge Centers • Re-usable Knowledge • Set-Based Design • Trade-Off Curves
3. Decide as late as possible:	• Just In Time Development • Late Decisions
4. Deliver as fast as possible	• Extreme Programming • Queue Sizing • Batch Sizing • Cycle Time • Queue Sizing • Takt Time
5. Empower the team	• Make Decisions At The Right Level • Self-Determination • Motivation
6. Build integrity in	• Conceptual Integrity • Customer Value • Perceived Integrity
7. See the whole	• System Thinking • Sub-optimization

Table 4: Lean Thinking Principles and Their Components

Before beginning the discussion of the seven principles of lean thinking, additional information around lean thinking in general and its history is required. Womack, Jones and Ross (1991) introduced the world to the term “Lean Thinking”. They suggest that lean thinking started back with Henry Ford and his production line innovations. While lean thinking may have started with Ford, the basic lean thinking principles as they are known today evolved after World War II from a small Japanese automobile company named Toyota and one of their engineers named Taiichi Ohno. The guiding principle of lean thinking is the absolute elimination of waste (Poppendieck 2001).

The Toyota organization was a successful spinning and weaving company and Ohno already had knowledge about how to build a good product. Basic ideas, such as

stopping the weaving machine as soon as a flaw was detected, were applied to automotive manufacturing. Ohno insisted that each automobile part was inspected for defects the minute it was produced and gave the power to the inspector to stop the line when a defect was identified (Womack, Jones and Ross 1991).

In describing an organization that is not lean, Poppendieck (2002) discusses the way Sears centralized the eyeglass laboratory of the first half of the 20th century. It consisted of a long distribution channel that included centralized equipment, huge distribution centers and lengthy distribution channels to realize the economies of scale. These mass production techniques create a huge amount of work that does not add end user value. The shipment of eyeglasses to a factory for one hour of processing adds far more time to the overall process than the manufacturing of them. Sears had a practice of building up an inventory of orders, resulting in having to track these orders, numerous queries about these orders, and constant order changes.

Womack, Jones and Ross (1991) compare lean thinking to this kind of long distribution channel:

> The lean producer, by contrast, combines the advantages of craft and mass production, while avoiding the high cost of the former and the rigidity of the later… Lean production is 'lean' because it uses less of everything compared with mass production – half the human effort in the factory, half the manufacturing space, half the investment in tools, half the engineering hours to develop a new product in half the time. Also, it requires keeping far less than half the inventory on site, results in many fewer defects, and produces a greater and ever growing variety of products. (p. 13)

Lean thinking principles provide a different aspect on concepts most managers are taught in school. Jerry Kilpatrick (2003) provides a table with a comparison (see Table 5: Traditional and Lean Views of Development Concepts).

Development Concept	Traditional Organization	Lean Organization
Inventory	An asset, as defined by accounting terminology	A waste – ties up capital and increase processing lead-time
Ideal Economic Order Quantity and Batch Size	Very large – run large batch sizes to make up for process downtime	ONE – continuous efforts are made to reduce downtime to zero
People Utilization	All people must be busy all the time.	Because work is performed based directly upon customer demand, people might not be busy
Process Utilization	Use high-speed processes and run them all the time.	Processes need to only be designed to keep up with demand
Work Scheduling	Build products to forecast	Build products to demand
Labor Costs	Variable	Fixed
Work Groups	Traditional (functional) departments	Cross-functional teams
Accounting	By traditional Financial Accounting Standards Board	"Through-put" Accounting
Quality	Inspect/sort work at end of process to make sure we find all errors	Processes, products and services are designed to eliminate errors

Table 5: Traditional and Lean Views of Development Concepts (from Kilpatrick, 2003)

The previous paragraphs provide an overview of lean thinking and its origins in Toyota Manufacturing. With that understanding, each of the seven principles analyzed in this study are described in further detail below. Each component of each principle is not examined; rather, selected components that best demonstrate the intention of the principle are addressed.

Principle 1 - Eliminate waste

The core principle of lean thinking that all others are based on is eliminating waste. Ohno defines this waste as anything that does not create value for the customer (Poppendieck and Poppendieck 2003). Value is measured by customer perception -- it doesn't matter if it makes things easier for the manufacturer, if it improves the profits of the company, or if it looks good on somebody's resume. If when viewed by the customer it is not perceived as providing value to them, then it is waste (Womack and Jones, 1996).

Muda: Taiichi Ohno (1988) uses the Japanese word *muda* to identify waste and defines it as:

> …defects (in products), overproduction of goods not needed, inventories of goods awaiting further processing or consumption, unnecessary processing, unnecessary movement (of people), unnecessary transport (of goods) and waiting (by employees for process equipment to finish its work or on an upstream activity) (pp. 19-20).

Non-value Added Processing: Muda has become one of the buzz words for lean thinking. Supporters of these principles use muda on a regular basis to describe things that do not add value. When listening to a group of lean thinkers it is not unusual to hear them say something is "muda" when it is determined to add no value. Womack and Jones (1996) provide the following examples of muda:

- Mistakes that require rectification
- Production of items that nobody wants
- Processing steps that are not needed
- Movement of resources without purpose
- People waiting for deliverables
- Defective releases
- Goods and services which don't meet the needs of the customer

In addition to this definition of muda, Shigeo Shingo identifies seven types of manufacturing waste, including: inventory, extra processing, overproduction, transportation, waiting, motion and defects (Shingo 1981).

Value Stream Maps: One way to identify waste is to create a value stream. Value streams are defined by Womack and Jones (1996) as "… the set of all the specific actions

required to bring a specific product through the three critical management tasks of any business". They further define these tasks as problem-solving, information management and physical transformation.

Problem-solving – Taking a concept from design to engineering to production.

Information Management – The management of the flow of customer orders from order-taking to the delivery schedule to the actual delivery of the product.

Physical Transformation – The process by which raw materials are transformed into a finished product.

Analysis of these value streams will result in assigning each step in the value chain to one of the following three categories: (1) Steps that unambiguously create value. (2) Steps that create no value whatsoever, but are unavoidable with current technologies. (3) Steps that create no value at all and can be immediately eliminated (Womack and Jones 1996).

Wasted Steps: Documents, diagrams, and other artifacts that are produced during the product development are only considered waste if nobody uses them. If business policies or processes require that an engineer or developer create a report that is never used by another step in the value chain, then it is muda. Poppendieck (2001) states "The burden is on the artifact to prove not only that it adds value to the final product, but also that it is the most efficient way of achieving that value."

Principle 2 – Amplify Learning

The central idea behind this principle is to retain your knowledge and make a decision based upon that knowledge. This is so obvious, but is one of the hardest to

accomplish. There is a whole area of study that addresses this principle known as Knowledge Based Development (KBD). The premise is to document knowledge learned so it does not have to be recreated in the future and is accessible to whoever requires it (Kennedy 2003).

Concurrent Development and Set-Based Design: Amplifying knowledge is not just about retaining what you have learned, but using concepts such as set-based design, concurrent development and tradeoff curves in addition to retained knowledge to make data-driven decisions. Set-based design and concurrent development are ways to design and develop multiple solutions for a problem in parallel. Instead of trying to predict the future by choosing one solution at the beginning of development, the design/development team selects multiple solutions and eliminates the non-optimal ones at the appropriate point. Some are eliminated as soon as they are discussed, others in the design phase, and more in the early stages of development until finally a clear optimal choice can be made (Poppendieck and Poppendieck 2003). At first glance this may seem like waste, but the learning from each eliminated option can be leveraged into development of the remaining options. While there are arguments that set-based design and concurrent development still create waste, but in comparison, finding out three-quarters of the way through development that the choice you made during design is invalid and having to start over generates significantly more waste (Ward 2002).

Trade-off Curves: A *trade-off curve* is model used to make a data-based decision. They make visible the physics and economics of a product or process. They can also show the physical limitations of a solution and where the value points are (Ward 2002). An example of a trade-off curve is shown in Figure 1. This curve sketches the characteristic trade-off between exhaust gas back-pressure and noise level. The areas not

on the curve are the infeasible regions, while the points on the curve are actual solutions. With this curve a decision can be made as to the optimal level of back-pressure to achieve an acceptable level of noise.

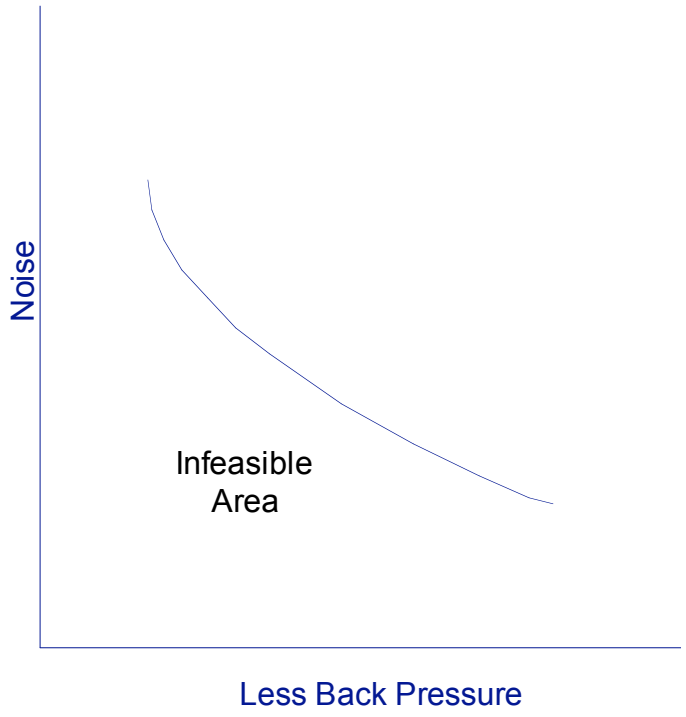

Figure 1: Trade-Off Curve Example

The core value of this principle is to retain knowledge and use that knowledge to make decisions based on data, not assumptions (Ward 2002).

Document Learning: Documentation of the process including cycle times, work flow and process definitions was created by the workers who performed the work, not desk-bound engineers (Womack, Jones and Ross 1991). This empowered the individuals and made them feel like part of the team. This team building also created almost a choreographed flow similar to a relay race team on the production line. As one process was completed, the baton was handed off to the next worker on the line. A culture where workers took pride and ownership in the entire product evolved resulting in an

environment where when one worker had to reset a machine or recover from some other malfunction, co-workers came to their aid.

Principle 3 - Decide as late as possible

The principle of deciding as late as possible is closely tied to set-based design and concurrent development. Windholtz (2003) says that the idea here is to wait to make any decision until the last possible moment and that moment is the point at which if you do not make a decision, an outside factor is going to make it for you. This ***late decision-making*** helps keep options open since if you have not made a choice, you still have multiple options. Also, if you have multiple options and one of them proves to be non-optimal, you do not have to realize the change cost that is associated with starting up a new option.

An example of this principle in today's market is provided by Poppendieck (2001):

> "In a market in which volatile technology requires constant product upgrades, Dell Computer has a huge advantage over its keenest competitors because it doesn't forecast demand; rather it responds to it by making-to-order in an average of 10 days. While Dell holds only days' worth of inventory, its competitors maintain weeks' worth. Dell's ability to make decisions as late as possible gives the company a significant competitive edge in a fast-moving market." (p. 4)

This is an excellent example of making a decision as late as possible. Dell waits until the last possible moment to buy components and then to build the PCs. This allows them to maintain minimum inventory of just what they need both in parts and finished product.

Enrico Zaninotto, an Italian economist, states that the underlying mechanism for controlling complexity in ***just-in-time systems*** **(JIT)** is minimizing irreversible actions (Poppendieck and Poppendieck 2003). Zaninoto also states that when a system that has options built into it is confronted by a system that has no options available, the system with the options will win in a complex market.

The final decision as to which solution to use is prolonged to the last possible moment, either when only one solution is obviously feasible or when a decision has to be made because other parts of the project are waiting on the solution. Delaying irreversible decisions until this last possible moment when uncertainty has been minimized leads to better decisions, limits risks, and provides value to the customer (Poppendieck and Poppendieck 2003).

Principle 4 - Deliver as fast as possible

The principle of 'deliver as fast as possible' complements 'decide as late as possible'. The less time between deliveries, the later you can make decisions. If it takes weeks instead of months to see the results of a change, you can make a decision about a change later in the process (Poppendieck and Poppendieck 2003).

Advantages of rapid delivery can also be seen in the requirements process. Windholtz (2003) states that the amount of change in requirements increase non-linearly as the length of the project increases. A typical 12-month process will generate a 25 percent change in requirements. However, the amount of average change over any given month is only 1–2 percent. By delivering more frequently, you can reduce the amount of change between any two deliveries during the development of a product.

Care must be taken when interpreting the meaning of the title of this principle. While it reads "deliver as fast as possible", it really means deliver as fast as is reasonable.

If a development team is told to deliver a product to test every morning, but it takes three days to perform the testing procedures and then it takes the developers two days to get fixes into a new product, then delivery on a daily basis does not add value. The idea behind this principle is to deliver on a schedule that optimizes development and provides usable deliveries for partners in the value chain (Reinerstsen 1997).

One way to determine the optimal time between deliveries is to utilize ***queuing theory***. A quick summary of how it works is provided by Poppendieck and Poppendieck (2003):

1. Measuring the amount of work waiting to be done is equivalent to measuring the ***cycle time*** of a system.
2. As variability increases (in arrival time or processing time), cycle time and work-in-queue will increase.
3. As ***batch size*** increases, variability in arrival and processing time increases and therefore cycle time and work-in-queue will increase.
4. As utilization increases, ***cycle time*** will increase non-linearly.
5. As variability increases, the non-linear cycle time happens at ever-lower levels of utilization.
6. Continuous flow requires a reduction in variability.
7. Variability may be reduced by an even arrival of demand, small batches and an even rate of processing and parallel processing.
8. Decreasing variability early in the process has larger impact than decreasing variability late in the process.

The application of queuing theory will enable an organization to optimize batch sizes and reduce cycle times resulting in a delivery schedule that is both fast and efficient.

Principle 5 - Empower the team

The quality of a team is the most important element in successfully delivering a product. If you make the team responsible for the outcome and give them the power to make decisions on how to make it happen, you have a group that is willing to take responsibility, motivated and acts like a team (Kilpatrick 2003).

Make Decisions at the Right Level: Empowering the team allows the decision-making process to be driven down to the lowest possible level in the organization (Womack, Jones and Ross 1991). The concept behind this principle is to allow the person who has the most knowledge about an issue to make the decision about how to resolve it. When there is a problem, a team of outside experts are not sent in to document what is wrong and how the process should be performed. People who know what is going on, the workers, are given the tools to evaluate and improve the process. Supervisors are trained to encourage work teams to solve their own problems (Poppendieck 2001).

An excellent example of the power of this principle is the General Motors plant in Freemont, California. GM closed it down in 1982 because of low production, low quality, high absenteeism and union problems. In 1984 it was reopened by the New United Motor Manufacturing, Inc. (NUMMI), a joint venture between GM and Toyota. Toyota managed the plant, but rehired eighty-five percent of the former employees including the entire union leadership. By 1986 the plant had the highest productivity rate of any GM plant, the lowest absenteeism rate and the substance abuse problems they had before the closure was now minimal. What happened? Management first taught employees how to design their own jobs then created small teams that designed their own work procedures and work standards with teams doing the same job on different shifts. Management and

engineers are there as tools for the teams to accomplish their jobs, not to dictate how they are done. Each team was given the responsibility for its own procedures, its own quality and for the smooth flow of parts from upstream to downstream teams (Poppendieck and Poppendieck 2003).

Motivation: Another way to empower a team is to give them purpose. People generally want to have a purpose in life and when they have that purpose they care about fulfilling it. Poppendieck and Poppendieck (2003) provide the six ways to help a team gain and hold a sense of purpose.

- Give them a clear and compelling purpose
- Be sure the purpose is achievable
- Give the team access to customers
- Let the team make its own commitments
- Management's role is to run interference
- Keep skeptics away from the team

Self-determination: These are only a few ways to aid a team in holding onto a purpose and empowering them to be accountable for themselves. If team members have a purpose and feel that they are responsible for it, then they will take ownership of the product they are producing.

In the Full Purpose section of Chapter 1. Purpose of the Study, the United States Marine Corp provides an example of the empowerment of the team principle. In the Marines decisions are made at the level that makes sense for the situation and time. During the high level planning stages of a mission, top level command are involved in the decision process of taking on a particular mission. During the next phase of the planning where it is decisions are made about when the operation will take place and who will be

implementing it, the next lower level of command is involved. This process proceeds down the command chain with the pretense that as the plan becomes more detailed it is developed by members closer to the implementation. Finally when the plan is actually executed, the soldiers that are on site and those on the front lines with the most current information are expected to make decisions and modify the plan as needed. This is an excellent example of driving the decision down to where it needs to me made (Poppendieck and Poppendieck 2003).

Principle 6 - Build integrity in

A study at Harvard Business School by Kim Clark (Clark and Fujimoto, 1994) shows that companies that value product integrity and make it a part of the development methodology consistently deliver superior products. Lean thinking takes this concept to heart and says that product integrity should influence everything you do when making decisions about the development of a product (Ward, 2002). Product integrity can be further defined by Poppendieck and Poppendieck (2003) as perceived and conceptual. They write:

> ***Perceived integrity*** means that the totality of the product achieves a balance of function, usability, reliability and economy that delights customers. ***Conceptual integrity*** means that the system's central concepts work together as a smooth cohesive whole. (p. 125).

The customer's whole experience with a product is its perceived integrity. This integrity can be influenced by how it's advertised, installed, how easy it is to use, is it updateable, affordability and many other factors (Poppendieck and Poppendieck 2003).

If a product has conceptual integrity, i.e. if its parts work together, the architecture has a balance between flexibility, maintainability, efficiency, and responsiveness. A product has to have conceptual integrity before it can achieve perceived integrity. If conceptual integrity has not been achieved, then there will be usability issues with the product. One of the major keys to integrity is to ensure that there is communication between the customer needs and the developers. While the developers may not need to talk directly to the customers they need to know what the ***customer values*** are so that they can make sure they are developing to those values.

Principle 7 - See the whole

A system is not just the sum of its parts, but rather the product of the interactions of these parts. Each piece of the system can be free of defects and meet all of its requirements, thusly perform perfectly. But if it does not interact correctly with the other parts of the system, the system is a failure. Usually all parts of the system are not developed by the same team and each team may lose sight of the overall goal of the whole system. It is important for not only the designers of the system, but also the individual developers of each of the system parts to understand and work toward the system goals (Poppendieck and Poppendieck 2003).

Traditional approaches of solving system problems can have negative impacts on the overall system. One of these approaches is known as limits to growth. A process that produces a desired result may create a secondary negative impact on the system. If the process that is producing the desired result is optimized to further improve on its results, the negative impact may also increase. A second approach is known as shifting the burden. In this practice the development team is aware of a defect in one part of the

system, but is not sure of how to address it. Instead of fixing the root cause of the problem, they compensate for it in another part of the system (Poppendieck and Poppendieck 2003).

Sub-optimization: Lean thinking has a tool known as the "five whys" to counter the results from limits to growth and shifting the burden. The five whys is a practice developed by Taiichi Ohno that involves asking "why" five times whenever a problem is encountered, in order to identify the root cause of the problem. The first "why" is asked at a high level and the answer generates an additional, more detailed "why". By the time the fifth "why" is reached, you are at the root cause of the problem (Womack and Jones 1996).

The Application of Lean Thinking Principles to the Software Development Process

Based on an understanding of lean thinking and the seven related principles, this section describes how to apply lean thinking principles to the software development process. Brief definitions of each of the seven principles of lean thinking are listed in Table 6: Lean Thinking Principles & Process Definitions.

Lean Thinking Principle	Process Definition
Eliminate waste	Only do things that provide customer perceived value.
Amplify learning	Retain knowledge so data driven decision can be made.
Decide as late as possible:	Wait until the last possible moment to make decisions.
Deliver as fast as possible	Deliver products as fast as make sense.
Empower the team	Decisions are made by the person who has the knowledge no matter what level of the organizati on they are in.
Build integrity in	Consider quality and integrity when making decisions.
See the whole	Make decisions based on the whole system, not individual components

Table 6: Lean Thinking Principles & Process Definitions

In Phase two of the data analysis, eleven sources are reviewed for the coexistence of discussion about lean thinking principles in relation to software development. In addition to the list of terms and phrases used in phase one, software development, software project management and software program management are included in the analysis. Any references to paradigms other than lean thinking are ignored since the focus of this research project is lean thinking and software development. The same format used to record results during phase one coding is utilized for phase two. As in phase one, the source and the page number are noted for further reference (see Appendix D: Phase Two Coding Results – Lean Thinking in Relation to Software Development). After the coding is completed, the identified processes are categorized according to the most appropriate lean thinking principle. The results from this categorizing are presented in Table 7: Lean Thinking Principles and Software Development Processes. To limit the scope of this paper, the data in this table is reviewed for the processes that would provide the most value in software development. These processes are identified in the table below in **bold** typeface.

Lean Thinking Principle	Software Development Processes
Eliminate waste	• **Identify Waste** • **Value Stream Mapping**
Amplify learning	• **Software Development Feedback Loops** • Iteration Planning • Synchronization • **Set-Based Development** • Trade Off Curves
Decide as late as possible:	• **Options Thinking** • **The Last Responsible Moment** • Depth-First Versus Breadth-First Problem Solving
Deliver as fast as possible	• Pull Systems • Queuing Theory • Cost Of Delay
Empower the team	• **Motivation** • Leadership • Expertise
Build integrity in	• **Perceived Integrity** • **Conceptual Integrity** • Refactoring • Testing
See the whole	• Measurements • Contracts

Table 7: Lean Thinking Principles and Software Development Processes

Some of the practices still considered standard for software development were abandoned long ago by other development disciplines. Set-based design, concurrent development, and late point decision making have not been generally considered as software development options since they are in direct contrast with the way software development has always been done. When organizations have tried to apply these principles to software, there have been mixed results at best. Poor results have partly been because of a naïve understanding of how these principles really work and a failure to recognize their limitations (Poppendieck and Poppendieck 2003).

In this section some of the principles of lean thinking discussed in the previous section are reviewed with a focus on software development. Principles such as ‘empower the team’, ‘build integrity in’ and ‘see the whole’ are

approached in the same manner in software development as in other product developments, and so are not reviewed again. Other principles such as 'eliminate waste' and 'build as fast as possible' provide some techniques that are unique to software development, and thus are reviewed further.

Eliminate waste in the software development process

Poppendieck and Poppendieck (2003) provide the following description of waste from Winston Royce. "… The fundamental steps of all software development are analysis and coding. While many additional development steps are required, none contribute as directly to the final product as analysis and coding and all drive up the development costs." (p. 4). From this can be interpreted that every step in software development is waste except coding and analysis. Poppendieck and Poppendieck (2003) identified seven wastes of software development and mapped them to Shingo's seven wastes of manufacturing presented in the previous section. This mapping is shown in Table 8: The Seven Wastes.

The Seven Wastes of Manufacturing	The Seven Wastes of Software Development
Inventory	Partially Done Work
Extra Processing	Extra Processes
Overproduction	Extra Features
Transportation	Task Switching
Waiting	Waiting
Motion	Motion
Defects	Defects

Table 8: The Seven Wastes (from Poppendieck and Poppendieck, 2003)

As with lean thinking in manufacturing, lean thinking in software begins with the ability to identify these wastes. The technique for accomplishing this in software development is the same as in manufacturing: create a value chain and then look for steps

in the value chain that add no customer perceived value and eliminate them (Poppendieck 2003).

Amplify Learning in the software development process

Software Development Feedback Loops: Just as in manufacturing, in software development it is important to make decisions based on data that is obtained from previous knowledge. Because of the serial and structured aspects of traditional software development, (see Table 1: Software Product Life Cycle), it is difficult to incorporate learning within them. A common software development paradigm is the waterfall model. The waterfall model is a sequential development model designed by Winston Royce in 1970. It makes the assumption that the details of a project are determined at the beginning of the development cycle (Gray and Larson 2000). In the original model, only feedback from a previous level was used. The assumption with this model is that you can get everything right in a single pass.

One way to incorporate learning into the process is to incorporate feedback loops from prototype builds. Royce provides a recommendation to the original Waterfall model as shown in Figure 2: Royce's Waterfall Recommendation. Royce recommends managing risk by "doing a project twice" to learn from actual prototype development experience, in order to expose problems earlier before they have serious impact on the main development effort. This means that a separate development process starts early on during the project when the preliminary design is complete. This development process creates a

prototype that validates the design and provides feedback into subsequent steps of the process.

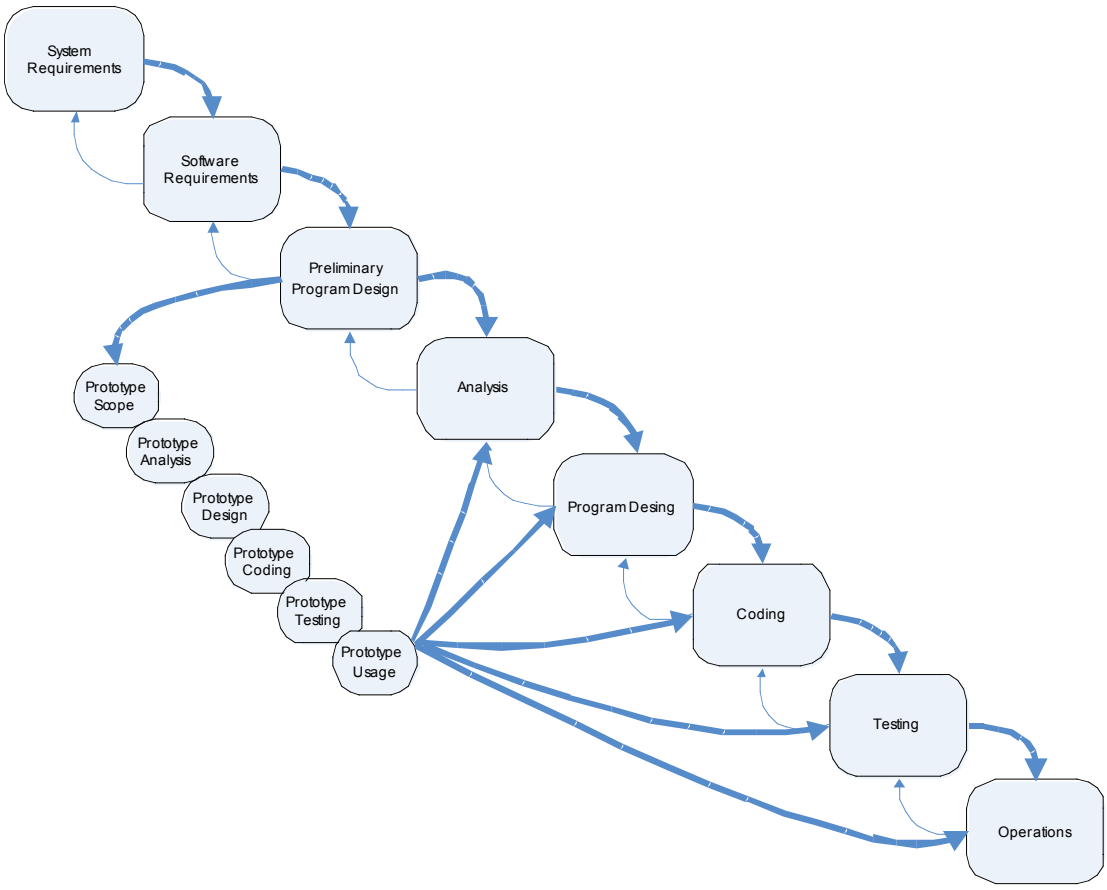

Figure 2: Royce's Waterfall Recommendation

Traditional software development incorporates a belief that an increase in feedback loops is a threat to the predetermined plan and that this feedback might cause changes to the design that would require looping back to a level already completed. Lean thinking supports the idea that increasing feedback is the single most effective way to deal with troubled programs. Listed below are some ways to increase feedback effectively, within the lean thinking process (Poppendieck and Poppendeick 2003):

- Run tests as soon as the code is written.
- Check out ideas by writing code.

- Gather more requirements from users an show them an assortment of potential screens to get their input.
- Study more carefully which tool to use and bring top candidates in-house and test them.

Set-based Development: Set-based development in software development is very similar to manufacturing. Set-based development involves looking at multiple solutions to a problem and developing several of them to determine which one is best. When it becomes apparent that an option is not going to provide the solution required, the development is terminated. The knowledge gained from the development of this option is transferred to the development of the remaining option(s). Windholtz (2003) provides an example of how set-based development can be applied to Web interface design.

> When we can't agree on how to structure the Web site, what we do is create two or three versions, with different paths and page layouts. We then do usability testing with several target users. It turns out that some features from each design are good and some are rather poor. We put together the best features of all the options and retest. Invariably we get a far better usability score with the combination. (p. 5)

Decide as late as possible in the software development process

Option Thinking: One method of implementing this principle is option thinking - a process in which you try and cover all bases early on and then make a choice at the last possible moment. Software development has traditionally been a predicative process, if product specifications are not right the first time, the cost of change is significant (Gray and Larson 2000). Agile software development uses option thinking in order to mitigate

the drawbacks of the predicative process. This new process allows decisions to be delayed until requirements and customer needs are more clearly understood (Poppendieck and Poppendieck 2003).

The Last Possible Moment: Concurrent software development, when tied with set-based development discussed previously in the "*amplify learning"* section, allows decisions to be made at the last possible moment. This does not equate to procrastination, but rather waiting until the moment at which failing to make a decision eliminates a possible alternative (Poppendieck 2001). Some tactics for late decision making are:

- Share partially complete design information.
- Organize for direct, worker-to-worker collaboration
- Develop a sense of how to absorb changes.
- Develop a sense of what is critically important in the domain.
- Develop of sense of when decision must be made.
- Develop a quick response capability.

Build integrity in to the software development process

Perceived Integrity: In traditional software development, perceived integrity is provided to the development via a series of documents in a serial process. These documents are translated into a new format. Additional information is added for the programmers two or three times as a result of customer interviews. The programmers may have no idea of the customers' view of system integrity and as a result they often have difficulty getting the software "right" in terms of customer needs (Poppendieck 2001).

One lean thinking principle that has not been discussed yet is the concept of a chief engineer. The chief engineer oversees the entire product development from the investigation to final release. They have understanding of customer needs, the technology required and the ramifications of trade-offs. They are responsible for the technical architecture of the product and how it is developed. Not all decisions are made by the chief engineer, but since they have the overall understanding of the product, they work with the other engineers to help them understand how the decisions they make will impact the integrity of the product (Poppendieck and Poppendieck 2003).

In software development there is also a need for a master developer who understands both the customer requirements and the type of tradeoffs the programmers might need to make. This master developer will facilitate the flow of information from the customer to the developers, resulting in a product that will have the system integrity the customer envisioned. This concept is not currently part of many software development organizations. They feel that is an overhead that they can't afford. In the absence of a master developer, Poppendieck and Poppendieck (2003) provide three techniques to establish information flow from the customer to the programmers:

- Small teams that have immediate access to the group that will determine the system's integrity. They should be developing smaller systems and delivering them on short iterations. These iterations should be reviewed often by a broad range of people who can evaluate the integrity when they see it.
- Feedback from customers performing tests result in excellent customer-programmer communication.
- Complex systems should be modeled in a language and format that can be understood by the customer, but provides enough detail for the programmer.

With these techniques, a master developer is still recommended for larger systems – someone who has a deep understanding of the customer and excellent technical skills. Their role is to facilitate the design process and represent the customer to the developers (Poppendieck and Poppendieck 2003).

Conceptual Integrity: As mentioned before there is a second type of integrity known as conceptual integrity. This addresses how a system's central concepts work together. The software architecture is the system structure and provides the desired features and functionality. This architecture is what defines the system's conceptual integrity. In traditional software development there is an expectation that the system will be perfect the first time. Everybody knows that while this might be an expectation, it is an unreal one. The reality is that software development is in a constant state of improvement of the solution, even after it has been delivered to the customer. Requirements change due to changes in technology, business domain and customer needs. A system with strong conceptual integrity will be adaptable to these changes (Poppendieck and Poppendieck 2003). Here are five techniques provided my Tom and Mary Poppendieck on maintaining conceptual integrity:

1. **Simplicity.** Whenever possible make the design and the implementation of the design as simple as possible.
2. **Clarify.** Develop code that is readable and easy to understand. Use naming conventions that are easily understood and are standard throughout the project. Even if it is clear to the programmer what the functionality of the code is, document it so any programmer can understand.
3. **Suitability for Use.** Every design has to accomplish its intended purpose. If a user-interface is not intuitive, it is not suitable. If performance has

degraded to an unacceptable level, address the issue no matter what changes are required.

4. **No Repetition**. If the same code exists in two different places, move it to a common area. This facilitates change and reduces risk of error by having the code in one place, you only have to change it in one place.
5. **No Extra Features.** If code is not needed, remove it. If code has become obsolete, remove it. Extra code in a system that is not being used anymore is waste.

Chapter V. Conclusion

While traditional project management methodologies were abandoned by other disciplines years ago, software development still utilizes them. Some of the reluctance to make the transition from these methodologies to lean development practices in software development is due to mixed results with forays in the past. These feelings of reluctance are a result of attempts to apply the processes and not the principles. Principles are guidelines, while processes are what you do to carry out the principles (Poppendieck and Poppendieck 2003).

A summary of the lean thinking principles discussed in relation to software development processes and the potential benefits that can be derived from these processes is presented in Table 9: Lean Thinking Principles, Software Development Processes and Potential Benefits.

Lean Thinking Principle	Software Development Processes	Potential Benefits
Eliminate Waste	Identify Waste	Eliminates non-value adding steps
	Value Stream Mapping	Eliminates non-value adding steps
Amplify Learning	Software Development Feedback Loops	Allows feedback to developers early in the development cycle
	Set-Based Development	Allows for the development of multiple options in parallel
Decide As Late As Possible	Options Thinking	Enables set-based development by examining all options
	The Last Responsible Moment	Enables the decision maker to have as much information as possible when making decisions, resulting in better decisions.
Build Integrity In	Perceived Integrity	Better customer satisfaction and a product that meets the customer need (more sales).
	Conceptual Integrity	Better product that requires less and easier maintenance.

Table 9: Thinking and Software Development Benefits

However, the results that can be realized from lean thinking principles are not free. Retaining knowledge and waste elimination takes time out of the engineer's already busy day. Empowering the team requires management to let go of decisions that they don't need to make or have the knowledge to make (Poppendieck and Poppendieck 2003).

Taking an organization from the traditional software development paradigm to one that incorporates the principles of lean thinking requires change in both technical procedures and organizational orientation. This transition requires having executive sponsorship and full support by all levels of the organization (Kennedy 2003). When a software development organization decides to implement lean thinking principles, they have to be careful to develop processes that promote lean thinking in their organizations. Processes that worked in company A might not work very well in company B, since lean thinking is as much a culture change as a process change (Womack and Jones 1996).

Lean thinking principles are guidelines about a discipline, not the processes around how to implement them. Principles are universal, while practices or processes are designed for specific organizations. Poppendieck and Poppendieck (2003) add to this with the statement "…there is no such thing as a best practice; practices must take context into account". Some software organizations have encountered this roadblock when attempting to implement lean thinking by using practices or processes that were successful in other organizations rather than the principles behind their success (Poppendieck and Poppendieck 2003).

This study presents data that offers an overview of the lean thinking principles in relation to traditional software development processes. Learning to identify waste is the core competency of any lean thinking application. The principles provide insight into

ways of identifying and minimizing waste. The ultimate goal of lean thinking is to create a system that has no waste, but in reality the best an organization can hope for is to eliminate as much waste as possible (Kilpatrick 2003).

Several methods are provided that will enable software development program managers to move their organizations into more efficient and thus more productive organizations. The most important aspects revealed in this study were:

- Eliminate processes, artifacts, overhead, etc. that do not provide customer perceived value (Womack and Jones 1996).
- Make decisions at the lowest possible level and at the last possible moment (Womack and Jones 1996).
- Develop solutions in iterative cycles, allowing feedback all the way through the development process, not just at the end (Poppendieck and Poppendieck 2003).
- Design and develop multiple solutions to a problem in parallel, combining best learning from failed options into the remaining option(s) (Womack and Jones 1996).
- Design and develop integrity into the system by understanding what the customer needs are and be flexible as these needs change (Poppendieck and Poppendieck 2003).

The integration of lean thinking principles into any product development process is not an easy one, but one that has proven to provide enormous benefits (Kennedy 2003). Taking that one step further and applying the principles to software development has additional challenges, but ones that are well worth the end result.

Jerry Kilpatrick (2003) provides the following reason for organizations to adopt lean thinking: "Lean organizations are able to be more responsive to market trends,

deliver products and services faster, and provide products and services less expensively than their non-lean counterparts. Lean crosses all industry boundaries, addresses all organizational functions, and impacts the entire system". (p. 5)

Engineers in traditional development organizations spend 20% of their time in value added activities. Kennedy (2003) states that engineers in organizations that utilize lean thinking principles spend 80% of their time on work that adds value. This improvement alone is enough to validate the advantages of lean thinking. In an article titled "Lean Programming" for Software Development Magazine, Mary Poppendieck (2003) summarizes the benefit of the application of lean thinking to software development: "Applied (lean thinking) to software program management as Lean Programming, these practices will lead to the highest quality, low cost, shortest lead-time software development possible." As demonstrated by this quote and in this study, lean thinking allows software development organizations to develop software in less time with fewer defects and with fewer resources. That alone should be enough to persuade any organization to make the change.

APPENDIX A

Definitions

Conceptual integrity - The system's central concepts work together as a smooth cohesive whole (Poppendieck and Poppendieck 2003).

Concurrent Development – The development of multiple solutions in parallel. As issues are identified that deem a solution invalid it is eliminated**.** (Poppendieck and Poppendieck 2003).

Conceptual Analysis – A method to establish the existence and frequency of concepts – most often represented by words of phrases – in a text. (CSU Writing Center 2005).

Content Analysis – A research tool used to determine the presence of certain words or concepts within texts or sets of text. (CSU Writing Center 2005)**.**

Cycle Time – The time required to complete one cycle of an operation. If cycle time for every operation in a complete process can be reduced to equal "takt time", products can be made in single-piece flow (Womack and Jones 1996).

Extreme Programming - A discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation. (Beck 2000)

Gantt Chart – A bar chart representation of a project schedule used during planning, resource scheduling and status reporting (Gray and Larson 2000).

Just-in-Time – (JIT) A system for producing and delivering the right items at the right time in the right amounts (Womack and Jones 1996).

Knowledge Based Development – (KBD) A system for developing new products that recognizes that all value is created from knowledge.

Lean Thinking – A way of thinking and a system to create future operational cycles and focuses on creating (re)usable knowledge. It contains principles such as concurrent development of multiple solutions, making decisions at the lowest possible level and using data to make decisions. (Ward 2002)

Muda – A Japanese word used to describe defects (in products), overproduction of goods not needed, inventories of goods awaiting further processing or consumption, unnecessary processing, unnecessary movement (of people), unnecessary transport (of goods) and waiting (by employees for process equipment to finish its work or on an upstream activity). (Ohno 1988))

Perceived Integrity – The totality of the product achieves a balance of function, usability, reliability and economy that delights customers (Poppendieck and Poppendieck 2003).

Pert Chart – A graphical representation of the flow of work in a project. (Gray and Larson 2000).

Project Management – The process of planning, organizing, staffing, directing and controlling the production of a system. (Gray and Larson 2000).

Set-Based Design – See Concurrent Development

Single-Piece Flow - A situation in which products proceed, one complete product at a time, through various operations in design, order-taking, and production, without interruptions, backflows or scarp. (Womack and Jones 1996).

Takt Time – The available production time divided by the rate of customer demand**.** (Womack and Jones 1996).

Trade Off Curve – A model that make visible the physics and economics of a product or process family, establishing fundamental limitations.

Value-Chain – The specific activities required to design, order and provide a specific product from concept to launch, order to delivery, and raw to materials into the hands of the customer (Womack and Jones 1996).

Waterfall Method – A sequential development model designed by Winston Royce in 1970. It makes the assumption that the details of a project are determined at the beginning of the development cycle (Gray and Larson 2000).

Work Breakdown Structures – A map of a project with varying levels of detail laid out in a hierarchical format (Gray and Larson 2000).

APPENDIX B

Phase One Coding Form

Source: _____

Term or Phrase	Page of Occurrence							

APPENDIX C

Phase One Coding Results – Terms that Define Lean Thinking

Included in this appendix is a sample of the phase one coding sheets, used to record results. To optimize space words, terms or phrases that were not found in the source are removed.

Source: Lean Thinking Banish Waste and Create Wealth in Your Corporation
James Womack and Daniel Jones

Term or Phrase	**Page of Occurrence**							
Customer Value	16	29	217	252				
Cycle Time	348	352						
Just In Time Development	126	208	231	236				
Lean Thinking	Multiple							
Muda	15	350	146	217	328	370		
Queue Sizing	351							
Takt Time	55	227	349	122				
Total Productive Maintenance	60	149	244					
Value Stream Maps	37	275	277	321	319			

Source: Managing the Design Factory
Donald G. Reinertsen

Term or Phrase	**Page of Occurrence**							
Batch Sizing	61	135	247	62	74			
Eliminate Waste	250							
Just In Time Development	1	51						
Non Value Added Processing	54							
Perceived Integrity								
Queue Sizing	42-67							
Trade-off Curves	210	30						
Wasted Steps	250							

Source: The Lean Development Skills Book
Allan C. Ward

Term or Phrase	Page of Occurrence							
Batch Sizing	63							
Concurrent Development	48	51						
Cycle Time	27	59						
Document Learnings	26							
Empower The Team	32							
Knowledge Centers	62							
Trade-off Curves	20	52	54					
Value Stream Maps	22							
Wasted Steps	23	26	27					

Source: Product Development for the Lean Enterprise
Michael Kennedy

Term or Phrase	Page of Occurrence							
Concurrent Development	16	33	75	121	168	247		
Cycle Time	232	234						
Knowledge Based Development	114	137	174	230	239	244		
Lean Development	120	229	242	244				
Lean Manufacturing	13	21	120	230	242			
Set-Based Design	199							

Source: IEE Lean Thinking

Term or Phrase	Page of Occurrence							
Batch Sizing	15							
Eliminate Waste		3	5					
Lean Thinking	1							
Takt Time	11	12						
Value Stream Maps	3	4	7	8	15			
Wasted Steps	2	3	5					

Source: Lean Principles
Jerry Kilpatrick

Term or Phrase	Page of Occurrence							
Batch Sizing	2	4						
Just In Time Development	1	4						
Lean Manufacturing	1							
Lean Thinking	1	2	5					
Non Value Added Processing	1							
Total Productive Maintenance	2							
Value Stream Maps	1	2						
Wasted Steps								

APPENDIX D

Phase Two Coding Results – Lean Thinking in Relation to Software Development

Included in this appendix is a sample of the phase two coding sheets, used to record results. To optimize space terms or phrases that were not found in the source are removed.

Source: _Lean Software Development An Agile Toolkit
Mary and Tom Poppendieck

Term or Phrase	Page of Occurrence					
Amplify Learning	22	27	32	34	42	
Batch Sizing	77					
Build Integrity In	125	127	135	142	145	148
Conceptual Integrity	127	135	152			
Concurrent Development	47	50	57			
Cycle Time	79					
Fast Delivery	69	88	85			
Eliminate Waste	2	5				
Empower The Team	97	107	114	142		
Late Decision	53	61	64			
Lean Thinking	9					
Perceived Integrity	126	129	134	139		
Queue Sizing	77	79	82			
Software Development	Too Many To Note					
Software Project Management	1					
Value Stream Maps	9	11				
Wasted Steps	4	6	7	9		

Source: Lean Programming Part 1 and 2
Mary Poppendieck

Term or Phrase	Page of Occurrence							
Concurrent Development	9							
Customer Value	5							
Cycle Time	2	4						
Eliminate Waste	3							
Empower The Team	3	7						
Extreme Programming (XP)	8							
Just In Time Development	2							
Lean Manufacturing	1							
Software Development	7							
Wasted Steps	2							

Source: Lean Software Development
Mark Windholtz

Term or Phrase	Page of Occurrence							
Customer Value	2							
Cycle Time	2							
Extreme Programming (XP)	2							
Lean Manufacturing	1							
Software Development	1	2						

Source: Lean Thinking The Theory Behind Agile Software Development
Mary Poppendieck

Term or Phrase	Page of Occurrence							
Customer Value	4	6	11					
Fast Delivery	7	13						
Eliminate Waste	5	10						
Empower The Team	15							
Late Decision	7							
Lean Manufacturing	2							
Set-Based Design	14							
Value Stream Maps	8							
Wasted Steps	10							

References

Astels, D., Miller, G and Noval, M. (2002) *A Practical Guide To Extreme Programming* Upper Saddle River, NJ: Prentice Hall PTR

Beck, K. (2000) *Extreme Programming Explained* Boston: Addison Wesley

Clark, Kim B., and Taskahiro Fujimoto (1994). *The Power of Product Integrity* Harvard Business School Press

Cochra, D., Kim, Y, Carl, H. and Weideman, M.(2001) [online; accessed 5/2004] Redesigning a Mass Manufacturing System to Achieve Today's Manufacturing System Objectives 2001 at: http://www.sysdesign.org/pdf/paper18.pdf

Cockburn, A. (2002) *Agile Software Development* Boston: Addison Wesley

Cusumano, M. and Nobeoka, K. (1998) *Thinking Beyond Lean* New York: The Free Press

CSU Writing Center [online; accessed 3/2005] Introduction to Content Analysis 2005 at: http://writing.colostate.edu/references/research/content/

Gray, C. and Larson, E. (2000) *Project Management The Managerial Process* Boston: Irwin McGraw-Hill

The IEE (2005) [online accessed 3/2005] Lean Thinking at http://www.iee.org/oncomms/pn/manufacturing/leanvillage/index.cfm

Jeffires, R (2001) [online; accessed 1/2005] What is Extreme Programming 2001 at http://www.xprogramming.com/xpmag/whatisxp.htm

Johnson, J. (2002) [online; accessed 4.2005] ROI – It's Your Job at: http://www.xp2003.org/talksinfo/johnson.pdf

Kennedy M. (2003) *Product Development for the Lean Enterprise* Richmond: The Oaklea Press

Kilpatrick, J (2003) [online accessed 3/2005] Lean Principles at: http://www.mep.org/textfiles/LeanPrinciples.pdf

Krippendorff, K. (2004) *Content Analysis An Introduction to Its Methodology* Thousand Oaks Sage Publications

Leedy, Paul and Ormrod, J (2001) *Practical Research Planning and Design* Upper Saddle River: Merrill Prentice Hall

Lian, Y and Ladeghem, H (2002) [online accessed 3/2005] An Application of Simulation and Value Stream Mapping in Lean Manufacturing at http://www.scs-europe.org/services/ess2002/PDF/log-11.pdf

Moore, G. (1991) *Crossing the Chasm* New York: Harper Business

Ohno, T (1988) *The Toyota Production System: Beyond Large Scale Production* Portland: Productivity Press

Poppendieck, M (June 2001) [accessed online 4/2005] Lean Programming Part 1 at: http://www.sdmagazine.com/documents/s=731/sdm0105e/0105e.htm

Poppendieck, M (June 2001) [accessed online 4/2005] Lean Programming Part 2 at: http://www.sdmagazine.com/documents/s=730/sdm0106g/

Poppendieck, M and Poppendieck T (2003) *Lean Software Development An Agile Toolkit* Boston: Addison Wesley

Poppendieck, M (2002) [online accessed 3/2005] Principles of Lean Thinking at: http://www.poppendieck.com/papers/LeanThinking.pdf

Poppendieck, M. [accessed online 3/2005] Lean Software Development C++ Magazine Methodology Issue Fall 2003 at: http://www.poppendieck.com/pdfs/Lean_Software_Development.pdf

Poppendieck, T. (2003) [online accessed 3/2005] The Agile Customer's Toolkit at: http://www.poppendieck.com/pdfs/Agile_Customers_Toolkit_Paper.pdf

Reinersten, D. (1997) *Managing the Design Factory* New York: The Free Press

Saloner, G., Shepard, A. and Podolny, J (2001) *Strategic Management* New York: John Wiley and Sons, Inc.

Shingo, S. (1981) *Study of "Toyota" Production System from Industrial Engineering Viewpoint.* Tokyo: Japan Management Association

Smith, P and Reinertsen D. (1998) *Developing Products in Half the Time* New York: John Wiley and Sons

University of Oregon [accessed online 3/2005) Evaluating Information on the World Wide Web service of university libraries at: http://libweb.uoregon.edu/guides/searchweb/evaluating.html

Ward A. (2002) *The Lean Development Skills Book* Ann Arobt: Dollar Bill Books

Windholtz, M. (2003) [accessed on line 3/2005] Lean Software Development at: http://www.objectwind.com/papers/LeanSoftwareDevelopment.html

Womack, J and Jones D. (1996) *Lean Thinking Banish Waste and Create Wealth In Your Corporation* New York: The Free Press

Womack, J, Jones, D. and Ross, D. (1991) *The Machine That Changed the World* Harper Perennial